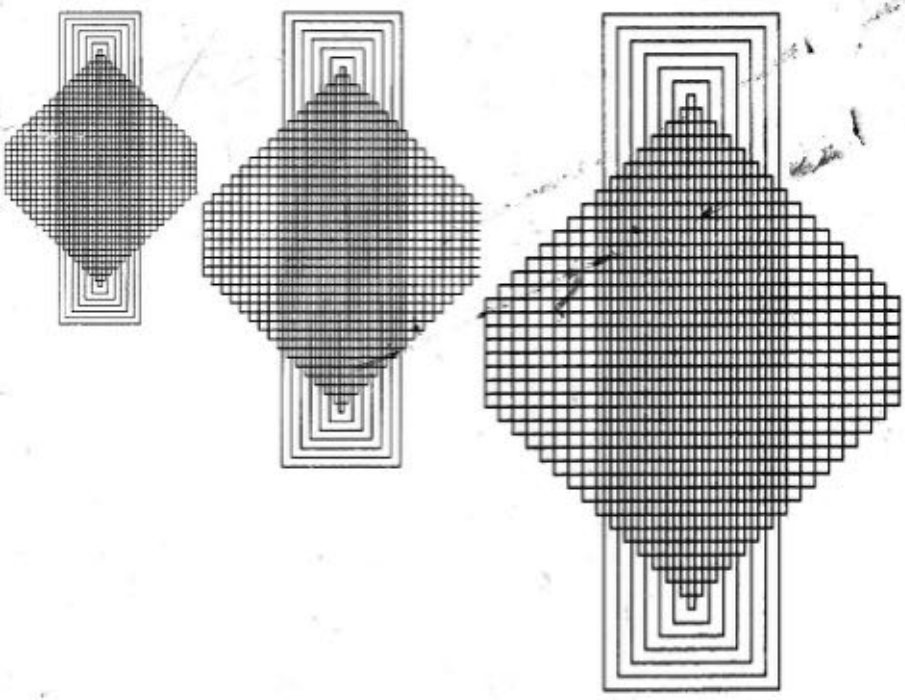
65

ATARI

ST BASIC™

Quick Reference Guide

For the advanced programmer.



Every effort has been made to ensure the accuracy of the product documentation in this booklet. However, because we are constantly improving and updating our computer software and hardware, Atari Corporation is unable to guarantee the accuracy of printed material after the date of publication and disclaims liability for changes, errors, and omissions.

ATARI, ST, ST BASIC, and TOS are trademarks or registered trademarks of Atari Corporation.
GEM is a registered trademark of Digital Research Inc.

Software copyright © 1986, Atari Corporation and Metacomco plc. All rights reserved.

No reproduction of this document or any portion of its contents is allowed without the specific written permission of Atari Corporation.

Copyright © 1987, Atari Corporation
Sunnyvale, CA 94086
All rights reserved.

TABLE OF CONTENTS

Welcome to Enhanced ST BASIC 1
- How to Use This Guide 1
- What's On the ST Language Disk 1
- The ST BASIC Sourcebook and Tutorial 2

Getting Started 2
- Loading ST BASIC 2

Converting Programs to ST BASIC 3
- Reading a BASIC Program 3
- Areas of Difference 3
  - ST BASIC 3
  - Non-ST BASICs 4

Error Codes and Messages 8

Quick Reference 9
- Declaration Characters 9
- Delimiters 9
- Editing Commands 9
- Operators 10
- Draw Mode 10
- Fill Pattern Style 10
- Line Pattern Style 11
- Ports 11
- Windows 11
- Sound 11

Command Listing 12

Statement Listing 12

Function Listing 13

System Variable Listing 13

Commands, Statements, Functions, and System Variables 14

Customer Support 29

Welcome to Enhanced ST BASIC™

This enhanced version of ST BASIC replaces the original ST BASIC provided with all ST™ Computers. Both versions are similar to standard BASIC dialects, but they use the windows, drop-down menus, and graphic elements of the GEM™ desktop. They also take advantage of the speed and graphic capabilities of the ST Computer system.

The new ST BASIC runs about three times faster than the original version, and performs more functions—with 33 new reserved words, an enlarged valid integer range, and more efficient syntax. The error message list has been expanded, and error messages now return clear explanations at each occurrence.

Enhanced ST BASIC is compatible with the original release of ST BASIC, so programs written in the earlier version can be used with the enhanced version of the language. Refer to the section, **Converting Programs to ST BASIC**, for information on how to use your old programs with the new version of the language.

How to Use This Guide

This guide is designed for advanced programmers who understand the BASIC language and are familiar with the standard procedures of the GEM desktop. The Quick Reference Guide is arranged so the programmer can understand the difference between this BASIC and the preceding version of ST BASIC. Also, the unique features of BASIC on the ST Computer are presented, along with demonstrations of how programs in other dialects of BASIC can be loaded and run with enhanced ST BASIC.

What's On the ST Language Disk

The ST Language disk that comes with your ST Computer contains the files necessary to run enhanced ST BASIC.

BASIC.PRG is the BASIC program.
BASIC.RSC is the resource file for the language.

Note: Disregard any other files that may be on the disk. Other files may hold the ST Desk Accessories or applications programs. However, only the two files listed above are necessary to program with enhanced ST BASIC.

The ST BASIC Sourcebook and Tutorial

The *ST BASIC Sourcebook and Tutorial* is the complete guide to enhanced ST BASIC. The new Sourcebook is 322 pages long, providing easy access to all levels of programming information. The beginning programmer is given an extensive tutorial, while the advanced programmer has access to complete technical documentation.

If you are interested in programming with enhanced ST BASIC, contact your Atari dealer and ask for purchase information on *The ST BASIC Sourcebook and Tutorial* (C026220 Rev B).

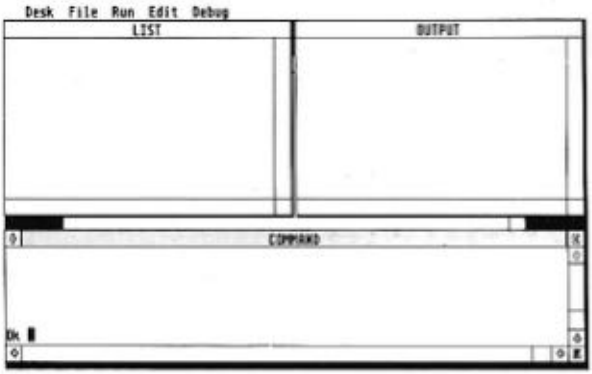
Getting Started

Before beginning with ST BASIC, make a backup copy of the ST Language disk. This will provide security against accidental damage to the original disk. (See the *ST Computer Owner's Manual* for instructions.)

After backing up your ST Language disk, you are ready to use ST BASIC. Begin by loading the language program into your ST Computer, following the instructions below.

Loading ST BASIC

1. With the ST Computer turned on and the GEM desktop displayed, double-click on the Floppy Disk A icon.
2. When the disk directory appears, double-click on BASIC.PRG. The ST BASIC desktop appears.



This desktop is your ST BASIC programming environment.

Note: ST BASIC uses the standard operating procedures of the GEM desktop for accessing menu items, selecting options, manipulating windows, and loading applications. Those procedures are explained in detail in the *ST Computer Owner's Manual*.

Converting Programs to ST BASIC

This section describes the improvements to ST BASIC that are included in enhanced ST BASIC, and the major differences between ST BASIC and non-ST BASICs. Use the information given here to convert programs written in another dialect of BASIC to run in ST BASIC.

Reading a BASIC Program

ST BASIC will only read programs that have been saved in ASCII text form. Make sure that programs you wish to convert are in ASCII text form.

An ST BASIC statement line must start with a line number, end with a line feed, and be no more than 255 characters long. Valid line numbers range from 1 to 65529 (0 to 65529 in other BASICs). ST BASIC does not recognize line continuation characters (line feed in some BASICs); the second part of any line continued in this way will be lost.

With the above exceptions, ST BASIC will preserve the text of your program even if it reports syntax errors; you can then use the ST BASIC Editor to change your program. The easiest way to convert a BASIC program is to LOAD it into ST BASIC, see what errors are reported, and EDIT the lines in place.

Areas of Difference

The differences between the enhanced ST BASIC and other BASICs are discussed in the following information.

ST BASIC

Enhanced ST BASIC is compatible with the earlier release of ST BASIC. Programs written in the earlier version can be used with the new release, if the following differences are taken into consideration.

DEF SEG has been eliminated. Instead, there are special versions of PEEK and POKE for word, byte, and long. These are PEEK_W, PEEK_B, PEEK_L, and POKE_W, POKE_B, and POKE_L. Programs using DEF SEG in old ST BASIC must be rewritten.

Addresses for PEEK and POKE use integers so that accuracy is sufficient.

Integers are now 32-bit numbers. This enlarges the valid range for integers, previously -32768 to 32767. The range is now from -2147483648 to 2147483647.

A new syntax for GEMSYS and VDISYS that works more efficiently than the old syntax has been introduced. The old syntax will still work with one addition: the number in parentheses must be placed in GEM_CONTRL(0) or GEMSYS or CONTRL(0) for VDISYS. Programs using VDISYS and GEMSYS should be modified to use the new syntax.

New reserved words have been added. These are:

AREA	GEM_ADDROUT	PATTERN
ASK MOUSE	GEM_CONTRL	PEEK_B
ASK RGB	GEM_GLOBAL	PEEK_L
BIOS	GEM_INTIN	PEEK_W
BOX	GEM_INTOUT	POKE_B
CLEAR	GEMDOS	POKE_L
DRAW	GSHAPE	POKE_W
DRAWMODE	LINEPAT	RGB
ED	MAT AREA	SSHAPE
ERR$	MAT DRAW	STATUS
GEM_ADDRIN	MAT SOUND	XBIOS

Programs written in the earlier ST BASIC that use any of those words in any way other than as reserved words must be rewritten.

Any syntax that calls for a list of x,y pairs is documented as requiring a semicolon (;) between the pairs. The semicolon enhances readability, although the old syntax will still work.

SYSTAB is now an array of 2-byte integers. Accesses to SYSTAB are as array elements whose index is half of the previous offset: for example, SYSTAB + 6 becomes SYSTAB(3).

INP used with -1 will not always return a negative number; however, it will return a non-zero number.

The period (.) is no longer allowed in variable names and keywords, and should be replaced with an underscore character.

Non-ST BASICs

Identifiers

Variable names and keywords in ST BASIC must start with a letter, and may contain letters (A to Z or a to z), digits (0 to 9), and the underscore character (_). Note that uppercase and lowercase letters are equivalent. Other BASICs allow the period character (.) but not the underscore, and in some instances the case of letters is significant.

String Constants

ST BASIC allows you to insert a double quote (") in a string constant by escaping it with another double quote. For example:

```
PRINT """This is truly a lemon," he said."
```

prints

```
"This is truly a lemon," he said.
```

Other BASICs treat the double quote as ending one string constant and starting another. Since separators between items in PRINT lists are not mandatory, you may find that a statement such as

```
PRINT "A""B"
```

in an existing BASIC program won't do what you expect in ST BASIC. ST BASIC prints "A""B" as A"B; other BASICs print it as AB.

Arithmetic

ST BASIC recognizes three numeric types: integer, single-precision floating point, and double-precision floating point.

Integers occupy 4 bytes and can represent numbers in the range -2147483648 to 2147483647. In many other BASICs, integers occupy 2 bytes and can represent numbers in the range -32768 to 32767. This affects the MKI$ and CVI functions and will affect the formats of records containing integers.

ST BASIC evaluates expressions involving only integer terms in integer, which may mean that intermediate values can overflow. For example:

```
a% = b% * c% - 170000
```

is evaluated in integer.

Floating point values are held in IEEE format. This has a slightly different dynamic range and precision than some other formats. ST BASIC does not recognize denormalized numbers, NANs, or infinities. For example:

```
PRINT 1E-38
```

will print 0, as 1E-38 is smaller than the smallest single-precision normalized number in IEEE format.

Mixed-mode arithmetic is carried out in double-precision because the precision of integers (31 bits plus sign) is greater than that of single-precision floating point (24 bits plus sign).

Translation and Interpretation

ST BASIC translates a program into a set of internal codes as you type it in or LOAD it. When a program is RUN, the internal code is executed. This differs from most microcomputer BASICs which translate and execute as they go, and causes differences in the effect of declarative statements.

DEF-type statements (which define default variable types) alter the action of the translator. They take effect as soon as they are typed in, no matter where in the program they happen to be. For that reason, the following example is not possible in ST BASIC:

```
100 input "what type ";a%
110 if a% = 1 then gosub 2000: goto 500
130 a = 5.0 : goto 600
500 a = "yes"
600 print a
700 end
2000 defstr a
2010 return
```

DEF FN defines a user function wherever it is placed in the program. The DEF FN function does not have to be executed to define the function. You cannot have two DEF FN statements defining the same function. For that reason, the following example is not possible in ST BASIC:

```
100 input "which def "; a%
110 if a% = 1 then gosub 2000 else gosub 2100
120 print FNR(1,2)
130 end
2000 def fnr(x,y) = x/y
2010 return
2100 def fnr(x,y) = y/x
2110 return
```

FOR/NEXT Restrictions

ST BASIC requires that each FOR statement be matched with exactly one NEXT statement, and each WHILE statement be matched with exactly one WEND statement. This matching is carried out before the program is run. Many BASICs carry out the matching as the program is running, allowing constructs such as:

```
100 FOR i% = 1 TO 1000
  .
  .
  .
300 IF i% > 500 THEN NEXT
  .
  .
  .
500 NEXT
```

ST BASIC will report an error at line 300, since it cannot tell before the program is run whether or not the NEXT at line 300 will be matched with the FOR at line 100.

ST BASIC does not permit a jump into a FOR/NEXT or WHILE/WEND loop from outside the loop. For example, given

```
10 GOTO 200
100 FOR i% = 1 to 1000
  .
  .
  .
200 PRINT "Value of i% is "; i%
  .
  .
  .
300 NEXT
```

ST BASIC will report an error when asked to RUN the program. Other BASICs may run the program, and report an error such as "NEXT without FOR" at line 300.

Command Environment

ST BASIC makes a distinction between statements (for example, PRINT), which are part of a program but cannot alter the program itself, and commands, which are used to examine or alter a program, and which cannot be part of the program itself. Some other BASICs allow commands to be part of the program they operate on. ST BASIC will report an error if it comes across a command on a program (line-numbered) line. ST BASIC commands are listed on page 12.

Error Codes and Messages

The following error codes and error messages are used with ERL, ERR, ERR$, and ERROR. (For explanations of these reserved words, refer to the section **Commands, Statements, Functions, and System Variables**.)

Code	Message
0–1	Undefined error
2	Syntax error
3	RETURN without GOSUB
4	Out of data
5	Illegal function call
6	Overflow
7	Out of memory
8	Undefined line number
9	Subscript out of range
10	Duplicate definition
11	Division by zero
12	Illegal in immediate mode
13	Type mismatch
14	Undefined error
15	String too long
16	Expression too complex
17	CONT valid only in BREAK mode
18	Undefined user function
19	Undefined error
20	RESUME without error
21	Undefined error
22	Missing operand
23	Program line too long
24–49	Undefined error
50	Field overflow
51	Invalid record length
52	Invalid file number
53	File not found
54	Invalid file mode
55	File already open
56	Undefined error
57	Device I/O error
58	File exists
59	Unable to create a file
60	Undefined error
61	Disk full
62	End of file
63	Invalid record number
64	Invalid filename
65	Invalid character in program file
66	Direct statement in file
67	KILL failed
68	Device unavailable
69–92	Undefined error
93	Undefined segment
94	Protected file
95	Not a BASIC program
96–98	Undefined error
99	—Break—
100	Undefined error
101	Program too large
102	Undefined error
103	Invalid line number
104	Missing line number
105	Undefined error
106	Statement not found
107	Integer overflow
108	Redo from start
109	Stop
110	GOSUBs nested too deep
111	Invalid BLOAD file
112–200	Undefined error
201	Invalid option
202	Command not allowed here
203	Line number required
204	FOR without NEXT
205	NEXT without FOR
206	Comma missing
207	Parenthesis missing
208	Option base must be 0 or 1
209	Undefined error
210	WHILE without WEND
211	WEND without WHILE
212	Undefined error
213	Duplicate DEF FN
214	Invalid jump into loop
215	Duplicate line number
216	Duplicate label
217–220	Undefined error
221	System error #%u
222	Program not run
223	Too many FOR loops
224–230	Undefined error

Quick Reference

Declaration Characters

Character	Function
%	Integer-type declaration character
$	String-type declaration character
!	Single-precision type declaration character
#	Double-precision type declaration character

Delimiters

Character	Purpose
'	Delimits remarks
"	Delimits strings
;	Delimits prompts
,	Delimits arguments
:	Delimits statements

Editing Commands

Function Key	Option
[F1]	Insert space
[F2]	Delete character
[F3]	Insert line
[F4]	Delete line
[F5]	Page up
[F6]	Page down
[F7]	Load text
[F8]	Save text
[F9]	New buffer
[F10]	Exit Edit window
EDIT [Return]	Start edit

Operators

Arithmetic Operators	Purpose
+	Adds; concatenates strings
–	Subtracts; negates
*	Multiplies
/	Divides
\	Converts to integers and divides
^ or **	Exponentiates

Relational Operators	Meaning
=	Equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
<>	Not equal to

Logical Operators	Purpose
AND	Performs logical "and"
EQV	Tests equivalence
IMP	Tests implication
NOT	Negates expression
OR	Performs logical "or"
XOR	Performs exclusive "or"

Draw Mode

Number	Mode
1	Replace
2	Transparent
3	Exclusive OR
4	Reverse transparent

Fill Pattern Style

Number	Style
0	Hollow
1	Solid
2	Pattern
3	Hatch

Line Pattern Style

Number	Style
1	Solid
2	Long dash
3	Dot
4	Dash dot
5	Dash
6	Dash dot dot
7	Defined by user

Ports

Number	Port
0	Printer
1	Modem
2	Console
3	MIDI

Windows

Number	Window
0	Edit
1	List
2	Output
3	Command

Sound

Parameters	Description	Range
Duration	Time is 1/50 second before next sound	
Note	Controls pitch: note position in scale	1 to 12
Octave	Controls pitch: octave number	1 to 8
Voice	Sound channel number	1 to 3
Volume	Controls degree of loudness	0 (off) to 15 (max.)

Command Listing

AUTO	EDIT	LOAD	RUN	TRON
BREAK	ERA	MERGE	SAVE	UNBREAK
CONT	FOLLOW	NEW	STEP	UNFOLLOW
DELETE	LIST	RENUM	TRACE	UNTRACE
DIR	LLIST	REPLACE	TROFF	

Statement Listing

AREA	FILL	OUT
ASK MOUSE	FOR	PATTERN
ASK RGB	FULLW	PCIRCLE
BLOAD	GEMDOS	PELLIPSE
BOX	GET	PRINT[#]
BSAVE	GOSUB	PRINT USING
CALL	GOTO	PUT
CHAIN (MERGE)	GOTOXY	QUIT
CIRCLE	GSHAPE	RANDOMIZE
CLEAR	IF	READ
CLEARW	INPUT[#]	REM
CLOSE	KILL	RESET
CLOSEW	LET	RESTORE
COLOR	LINE INPUT[#]	RESUME
COMMON	LINEF	RETURN
DATA	LINEPAT	RGB
DEF FN	LPRINT	RSET
DEFDBL	LSET	SOUND
DEFINT	MAT AREA	SSHAPE
DEFSNG	MAT DRAW	STOP
DEFSTR	MAT LINEF	SWAP
DIM	MAT SOUND	SYSTEM
DRAW	NAME	WAVE
DRAWMODE	NEXT	WEND
ELLIPSE	ON	WHILE
END	ON ERROR GOTO	WIDTH
ERASE	OPEN	WRITE[#]
ERROR	OPENW	XBIOS
FIELD	OPTION BASE	

Function Listing

ABS
ASC
ATN
BIOS
CDBL
CHR$
CINT
COS
CSNG
CVD
CVI
CVS
EOF
ERR$
EPX
FIX
FLOAT
FRE
GEMSYS
HEX$
INP
INPUT$
INSTR
INT
LEFT$
LEN
LOC
LOF
LOG
LOG10
LPOS
MID$
MKD$
MKI$
MKS$
OCT$
PEEK
PEEK_B
PEEK_L
POKE
POKE_B
POKE_L
POS
RIGHT$
RND
SGN
SIN
SPACE$
SPC
SQR
STR$
STRING$
TAB
TAN
VAL
VARPTR
VDISYS

System Variable Listing

CONTRL
ERL
ERR
GEM_ADDRIN
GEM_ADDROUT
GEM_CONTRL
GEM_GLOBAL
GEM_INTIN
GEM_INTOUT
INTIN
INTOUT
PI
PTSIN
PTSOUT
STATUS
SYSTAB

Commands, Statements, Functions, and System Variables

Name	Format/Description
=	<*variable*> = <*numeric expression*> |"string" Assigns value to variable (see LET).
ABS	ABS (<*numeric expression*>) Returns absolute value of argument.
AREA	AREA <*point list*> Draws a filled polygon.
ASC	ASC (<*string expression*>) Returns ASCII code of specified character.
ASK MOUSE	ASK MOUSE <*x*>, <*y*>, <*b*> Returns the current mouse coordinates and button status into the specified variables.
ASK RGB	ASK RGB <*reg*>, <*r*>, <*g*>, <*b*> Places the actual red, green, and blue values of the specified palette into the listed variables.
ATN	ATN (<*numeric expression*>) Returns arctangent of argument.
AUTO	AUTO [<*starting line number*>] [,<*increment*>] Automatically numbers lines on entry.
BIOS	BIOS <*numeric expression*>, <*arglist*> Provides an operating system call to BIOS.
BLOAD	BLOAD <*filespec*>, <*address*> Loads specified binary file.
BOX	BOX [FILL] <*x1,y1*>;<*x2,y2*> Draws a box.
BREAK	BREAK [<*line number list*>] Inserts breaks between lines of executing program.
BSAVE	BSAVE <*filespec*>, <*address*>, <*length*> Saves a section of memory in a binary file.

CALL	CALL <*numeric variable*> [(<*parameter list*>)] Calls machine language subroutine.
CDBL	CDBL (<*numeric expression*>) Converts argument to double-precision number.
CHAIN	CHAIN <*filespec*> [,<*line descriptor*>] [,ALL] Replaces current program with specified program and executes it.
CHAIN MERGE	CHAIN MERGE <*filespec*> [,<*line descriptor*>] [,DELETE <*line descriptor list*>] Merges current program with specified program and executes result.
CHR$	CHR$ (<*numeric expression*>) Converts integers to one-character strings according to ASCII code.
CINT	CINT (<*numeric expression*>) Converts argument to integer.
CIRCLE	CIRCLE <*x*>, <*y*>, <*radius*> [,<*start angle,end angle*>] Draws circles and arcs.
CLEAR	CLEAR Sets all numeric variables to 0 and string variables to null. Undefines all arrays.
CLEARW	CLEARW <*numeric expression*> Clears ST BASIC windows.
CLOSE	CLOSE [#]<*file number*> Ends input and output and closes data file.
CLOSEW	CLOSEW <*window number*> Closes BASIC windows.
COLOR	COLOR [<*text color, fill color, line color, index, style*>] Sets text, fill, and plot colors and fill patterns.
COMMON	COMMON <*variable*> [,<*variable*> ...] Passes specified variables to a chained program.
CONT	CONT Resumes execution after a break.

CONTRL — CONTRL (<*offset*>) = <*expression*>
A VDI-related system variable that can act as a built-in array.

COS — COS (<*numeric expression*>)
Returns cosine of argument.

CSNG — CSNG (<*numeric expression*>)
Converts argument to single-precision number.

CVD — CVD (<*8-byte string*>)
Converts 8-byte string to double-precision number.

CVI — CVI (<*4-byte string*>)
Converts 4-byte string to integer.

CVS — CVS (<*4-byte string*>)
Converts 4-byte string to single-precision number.

DATA — DATA <*constant*> [,<*constant*> ...]
Provides data for a READ statement to use.

DEF FN — DEF FN <*function name*>
[(<*variable list*>)] = <*definition*>
Defines user function.

DEFDBL — DEFDBL <*letter*> [– <*letter*>]
Defines range of initials as double-precision numbers.

DEFINT — DEFINT <*letter*> [– <*letter*>]
Defines range of initials as integers.

DEFSNG — DEFSNG <*letter*> [– <*letter*>]
Defines range of initials as single-precision numbers.

DEFSTR — DEFSTR <*letter*> [– <*letter*>]
Defines range of initials as strings.

DELETE — DELETE <*line number*> [– <*line number*>]
Erases program lines from memory.

DIM — DIM <*array name*> (<*subscript*> [,<*subscript*>] ...) [,<*array name*> (<*subscript*>, <*subscript*> ...)]
Associates variable name with array of specified dimensions.

DIR — DIR [<*disk drive*>:] [[<*filename*>].[<*extension*>]]
Lists a directory.

DRAW — DRAW <*point list*>
Draws a line through the vertices given in the argument list.

DRAWMODE — DRAWMODE <*integer variable*>
Sets the current drawing mode.

EDIT — EDIT [<*line number*>]
Edits current program.

ELLIPSE — ELLIPSE <*x*>, <*y*>, <*horizontal radius*>, <*vertical radius*> [,<*start angle*>, <*end angle*>]
Draws an ellipse.

END — END
Ends program, closes files, and returns to immediate mode.

EOF — EOF (<*file number*>)
Detects end of file.

ERA — ERA [<*disk drive*:>] [<*filename*>]
Deletes a file from the disk.

ERASE — ERASE <*array name*> [,<*array name*>] ...
Erases arrays.

ERL — ERL = <*error line*>
Contains the line number where an error occurred.

ERR — ERR = <*error code*>
Contains an error code.

ERR$ — ERR$(<*n*>)
Returns an error message for the specified code.

ERROR — ERROR <*numeric expression*>
Simulates an error.

EXP — EXP (<*numeric expression*>)
Returns e to specified power.

FIELD — FIELD #<*file number*>, <*field width*> AS <*string variable*> [,<*field width*> AS <*string variable*>] ...
Allocates space for variable values in file buffer.

FILL — FILL <*x*>, <*y*>
Fills shapes with colors or patterns.

FIX — FIX (<*numeric expression*>)
Truncates argument to integer.

FLOAT — FLOAT (<*integer expression*>)
Converts an integer to a single-precision number.

FOLLOW — FOLLOW <*variable*> [,<*variable*> ...]
Tracks values of specified variables during program execution.

FOR ... TO — FOR <*counter variable*> = <*start*> TO <*limit*> [STEP<*increment*>]
Initiates program loop.

FRE — FRE [(<*numeric expression*>)]
Returns number of bytes in memory unused by ST BASIC.

FULLW — FULLW <*window number*>
Sets ST BASIC windows to full screen size.

GEM_ADDRIN — GEM_ADDRIN (<*offset*>) = <*expression*>
A GEM-related system variable that can act as a built-in array.

GEM_ADDROUT — GEM_ADDROUT (<*offset*>) = <*expression*>
A GEM-related system variable that can act as a built-in array.

GEM_CONTRL — GEM_CONTRL (<*offset*>) = <*expression*>
A GEM-related system variable that can act as a built-in array.

GEM_GLOBAL — GEM_GLOBAL (<*offset*>) = <*expression*>
A GEM-related system variable that can act as a built-in array.

GEM_INTIN — GEM_INTIN (<*offset*>) = <*expression*>
A GEM-related system variable that can act as a built-in array.

GEM_INTOUT — GEM_INTOUT (<*offset*>) = <*expression*>
A GEM-related system variable that can act as a built-in array.

GEMDOS — GEMDOS <*numeric expression*>, <*arglist*>
Provides an operating system call to GEMDOS.

GEMSYS — GEMSYS <*AES Op Code*>
Accesses the operating system's AES interface.

GET — GET[#] <*file number*> [,<*record number*>]
Reads record from random file into buffer.

GOSUB — GOSUB <*line descriptor*>
Sends program control to subroutine.

GOTO — GOTO <*line descriptor*>
Jumps program execution to specified line.

GOTOXY — GOTOXY <*column position*>, <*row position*>
Places output cursor at specified column and row position.

GSHAPE — GSHAPE <*x1*>, <*y1*>, <*array*>
Writes the raster stored in the specified array to the screen.

HEX$ — HEX$ (<*numeric expression*>)
Returns hexadecimal equivalent of argument in string form.

IF

IF <*relationship*> THEN
<*line number*> | <*label*>

IF <*relationship*> GOTO
<*line number*> | <*label*>

IF <*relationship*> THEN <*line number*>
[ELSE <*line number*> | <*label*>]

IF <*relationship*> THEN <*clause*>
[ELSE <*clause*>]

Performs condition and decides on the direction of program flow according to the result.

INP

INP (<*port number*>)

Returns a byte value from a selected port.

INPUT

INPUT [;] [<*prompt string*> <;|,>]
<*variable*> [, <*variable*>] ...

Enters keyboard input during program execution.

INPUT#

INPUT# <*file number*>, <*variable*>
[, <*variable*>] ...

Assigns sequential file data to specified variable(s).

INPUT$

INPUT$ (<*number of characters*> [,[#]
<*file number*>])

Returns string of specified length from keyboard or data file.

INSTR

INSTR ([<*starting point*>,]
<*target string*>, <*pattern string*>)

Searches for string within string and returns position.

INT

INT (<*numerical expression*>)

Rounds argument to next lower integer.

INTIN

INTIN (<*offset*>) = <*expression*>

A VDI-related system variable that can act as a built-in array.

INTOUT

INTOUT (<*offset*>) = <*expression*>

A VDI-related system variable that can act as a built-in array.

KILL

KILL <*string expression*>

Deletes a file.

LEFT$ LEFT$ (<*target string*>, <*number of characters*>)
Returns substring from beginning of target string.

LEN LEN (<*string expression*>)
Returns length of specified string.

LET LET <*variable*> = <*numeric expression*> | "string"
Assigns value to variable.

LINE INPUT LINE INPUT [[;]"prompt string"; |[;]"prompt string",] <*string variable*>
Assigns sequential file record to specified variable.

LINE INPUT# LINE INPUT# <*file number*>, <*string variable*>
Reads line from data file and assigns it to string variable.

LINEF LINEF <*point list*>
Draws a line through the vertices given in the argument list.

LINEPAT LINEPAT <*style*> [,<*pattern*>]
Sets the line pattern.

LIST LIST [<*line descriptor list*>]
Displays specific program line(s) in the List window.

LLIST LLIST [<*line descriptor list*>]
Prints specific program line(s) on the printer.

LOAD LOAD <*filename*>
Loads specified program.

LOC LOC (<*file number*> or <*record number*>)
Returns record number or number of characters read or written.

LOF LOF (<*file number*>)
Returns length of file.

LOG LOG (<*numeric expression*>)
Calculates and returns natural logarithm of argument.

LOG10 — LOG10 (<*numeric expression*>)
Calculates and returns base-10 logarithm of argument.

LPOS — LPOS [(<*dummy argument*>)]
Returns the position of the print-head.

LPRINT — LPRINT [<*list of expressions*>]
Prints data on the printer.

LPRINT USING <*format string expression*>; [<*list of expressions*>]
Prints data on printer using specified format.

LSET — LSET <*string variable*> = <*string expression*>
Moves data into string and left-justifies it.

MAT AREA — MAT AREA <*count*>, <*array*>
Draws a filled polygon, taking the vertices of the polygon from the specified array.

MAT DRAW — MAT DRAW <*count*>, <*array*>
Draws a line, taking the vertices of the line from the specified array.

MAT LINEF — MAT LINEF <*count*>, <*array*>
Draws a line, taking the vertices of the line from the specified array.

MAT SOUND — MAT SOUND <*array*>
Submits the specified array to the sound daemon.

MERGE — MERGE <*filename*>
Merges specified program with resident program.

MID$ — MID$ ("target string", <*start*> [,<*length*>])
Returns specified substring from target string.

MID$ — MID$ (<*string variable*>, <*start*> [,<*length*>]) = "string"
Substitutes one substring for another in existing string value.

MKD$ — MKD$ (<*numeric expression*>)
Converts double-precision number to string.

MKI$ MKI$ (<*integer*>)
Converts integer to string.

MKS$ MKS$ (<*numeric expression*>)
Converts single-precision number to string.

NAME NAME <*old filename*> AS <*new filename*>
Changes name of file.

NEW NEW [<*filename*>]
Clears memory for new program and optionally names the new program.

NEXT NEXT [<*counter*> [,<*counter*>]] ...
Defines end of loop.

OCT$ OCT$ (<*numeric expression*>)
Returns octal equivalent to argument in string form.

ON ON <*expression*> GOSUB <*line descriptor*> [,<*line descriptor*>] ...
Defines multiple branch to subroutines.

ON <*expression*> GOTO <*line descriptor*> [,<*line descriptor*>] ...
Defines multiple branch.

ON ERROR GOTO ON ERROR GOTO 0 | <*line descriptor*>
Defines starting line of error trap.

OPEN OPEN "mode", # <*file number*>, "filename" [,<*record length*>]
Opens specified data file.

OPENW OPENW <*window number*>
Opens ST BASIC windows.

OPTION BASE OPTION BASE <*0 | 1*>
Sets base for array dimensions.

OUT OUT <*port number*>, <*byte*>
Sends a byte value to a selected output port.

PATTERN PATTERN <*plane*>, <*array*>
Sets the fill style.

PCIRCLE

PCIRCLE <*x*>, <*y*>, <*radius*>
[,<*start angle*>, <*end angle*>]
Draws solid circles and pie shapes.

PEEK

PEEK_B (<*address*>)
Returns an 8-bit value at memory address.

PEEK (<*address*>)
Returns a 16-bit value at memory address.

PEEK_L (<*address*>)
Returns a 32-bit value at memory address.

PELLIPSE

PELLIPSE <*x*>, <*y*>, <*horizontal radius*>,
<*vertical radius*>, <*start angle*>, <*end angle*>
Draws a solid ellipse or elliptical pie shape.

PI

PI = <*variable*>
Holds the value of pi.

POKE

POKE_B (<*address*>, <*data*>)
Inserts an 8-bit value at memory address.

POKE (<*address*>, <*data*>)
Inserts a 16-bit value at memory address.

POKE_L (<*address*>, <*data*>)
Inserts a 32-bit value at memory address.

POS

POS (<*file number*>)
Returns number of characters printed since last new line, or the current position of the cursor on the screen or printer.

PRINT

PRINT [<*print item*>] <;|,>
[<*print item*> [<;|,>] ...]]

?[<*print item*>] <;|,> [<*print item*>
[<;|,>] ...]]
Displays data on the screen.

PRINT USING

PRINT USING <*"format string"*>;
<*variable list*>
Prints data on screen using specified format.

PRINT# <*file number*>, USING
<*"format string"*>; <*expression*>
[,<*expression*> ...]
Prints data to file using specified format.

PRINT# PRINT# <*file number*>, <*print item*> [<*print item*> ...]
Outputs data to file.

PTSIN PTSIN (<*offset*>) = (expression)
A VDI-related system variable that can act as a built-in array.

PTSOUT PTSOUT (<*offset*>) = (expression)
A VDI-related system variable that can act as a built-in array.

PUT PUT [#] <*file number*> [,<*record number*>]
Writes record to random access file.

QUIT QUIT
Leaves ST BASIC and returns to the GEM desktop.

RANDOMIZE RANDOMIZE [<*numeric expression*>]
Seeds random number generator.

READ READ <*variable*>, <*variable*> ...
Assigns item(s) from DATA statement to specified variable(s).

REM REM <*remark*>
' <*remark*>
Inserts remark in program.

RENUM RENUM [<*new first line*>] [,<*increment*>]
Renumbers current program lines.

REPLACE REPLACE [<*filename*>] [,<*line number list*>]
Replaces existing program with new version.

RESET RESET
Places the contents of the Output window into the graphics buffer.

RESTORE RESTORE [<*line descriptor*>]
Resets pointer to specified DATA statement.

RESUME RESUME [NEXT|0|<*line descriptor*>]
Defines point of return from error trap.

RETURN RETURN
Marks end of a subroutine.

RGB RGB <*reg*>, <*r*>, <*g*>, <*b*>
Sets color palette to the specified red, green, and blue proportions.

RIGHT$ RIGHT$ (<*target string*>, <*integer*>)
Returns substring from end of target string.

RND RND [(<*numeric expression*>)]
Returns random number.

RSET RSET <*string variable*> = <*string expression*>
Moves data into string and right-justifies it.

RUN RUN [<*filename*>] [,<*line descriptor*>]
Executes program.

SAVE SAVE [<*filename*>] [,<*line descriptor list*>]
Saves the current program in source form.

SGN SGN [(<*numeric expression*>)]
Returns the sign of a number.

SIN SIN (<*numeric expression*>)
Returns the sine of the argument.

SOUND SOUND <*voice*>, <*volume*>, <*note*>, <*octave*>, <*duration*>
Makes musical notes.

SPACE$ SPACE$ (<*numeric expression*>)
Returns string of specified length filled with spaces.

SPC PRINT SPC (<*numeric expression*>)
Inserts specified number of blank spaces in print string.

SQR SQR (<*numeric expression*>)
Returns square root of argument.

SSHAPE SSHAPE <*x1,y1*>; <*x2,y2*>, <*array*>
Saves a raster in a specified array.

STATUS — STATUS = <*variable*>
Holds the value returned from each call to TOS™, GEM, VDI, or AES.

STEP — STEP [<*filename*>]
[,<*line descriptor number*>]
Executes program line by line.

STOP — STOP
Halts program execution.

STR$ — STR$ (<*numeric expression*>)
Converts numeric argument to a string.

STRING$ — STRING$ (<*numeric expression*>,
<*numeric expression*> | <*string expression*>)
Returns string of given length filled with specified character.

SWAP — SWAP <*first variable*>, <*second variable*>
Exchanges values of specified variables.

SYSTAB — SYSTAB (<*offset*>) = <*expression*>
Provides a system pointer table.

SYSTEM — SYSTEM
Exits ST BASIC and returns to the GEM desktop.

TAB — PRINT TAB (<*tab position*>)
Inserts tabs in PRINT statement.

TAN — TAN (<*angle in radians*>)
Returns tangent of argument.

TRACE — TRACE [<*line number*> – <*line number*>]
Steps through execution printing specified lines.

TROFF — TROFF [<*line number*> – <*line number*>]
Turns off TRON.

TRON — TRON [<*line number*> – <*line number*>]
Steps through execution printing specified lines and variable values.

UNBREAK — UNBREAK [<*line number*> – <*line number*>]
Turns off BREAK.

UNFOLLOW	UNFOLLOW [<*variable*>[, <*variable*>] ...] Turns off FOLLOW.
UNTRACE	UNTRACE [<*line number*> – <*line number*>] Turns off TRACE.
VAL	VAL (<*string expression*>) Returns numeric value of specified string.
VARPTR	VARPTR (<*variable*> | #<*file number*>) Returns offset of parameter from heap segment.
VDISYS	VDISYS [(dummy argument)] Allows the user access to the operating system's VDI interface.
WAVE	WAVE <*enable*>, <*envelope*>, <*shape*>, <*period*>, <*delay*> Controls the waveforms used in SOUND statements.
WEND	WEND Defines end of WHILE.
WHILE	WHILE <*logical expression*> Defines start and condition of indefinite loop.
WIDTH	WIDTH [#<*file number*>,] <*width*> Sets width for screen output. WIDTH LPRINT <*width*> Sets width for printer output.
WRITE	WRITE [<*expression*>] [,<*expression*>] Displays output on the screen.
WRITE#	WRITE# <*file number*>, <*expression*> [,<*expression*>] ... Sends output to sequential file.
XBIOS	XBIOS <*function*> [,<*arglist*>] Provides an operating system call to XBIOS.

Note: <*relationship*> is a numeric expression which evaluates to an integer value. If the value is zero, it is taken to be false; if it is non-zero, it is taken to be true.

CUSTOMER SUPPORT

Atari Corporation welcomes questions about your ATARI Computer products. Write to:

Atari Corporation
Customer Relations
P.O. Box 61657
Sunnyvale, CA 94088

Please write the subject of your letter on the outside of the envelope.

ATARI User Groups are outstanding sources of information on how to get the most from your ATARI Computer. To receive a list of ATARI User Groups in your area, send a self-addressed, stamped envelope to:

Atari Corporation
User Group List
P.O. Box 61657
Sunnyvale, CA 94088

Interested in programming with enhanced ST BASIC? The new *ST BASIC Sourcebook and Tutorial* is the resource you need.

This 322-page manual includes a complete tutorial for the beginning programmer, plus an exhaustive reference section for the advanced BASIC programmer. The *ST BASIC Sourcebook and Tutorial* is the only documentation you will ever need to take advantage of this new BASIC language.

For purchase information, contact your Atari dealer and ask for *The ST BASIC Sourcebook and Tutorial* (C026220 Rev B).

ATARI®

Copyright © 1987 Atari Corporation
Sunnyvale, CA 94086
All rights reserved.

C100536-001 REVA
C033007-0B1
Printed in Taiwan
1988 1 C C